National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-983

A Semi-Automatic System for Document Preparation

P. Chartier and F. Blais

May 1986

Canada

NRCC No. 26786

86-06466/

Copyright 1986 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1986 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

A SEMI-AUTOMATIC SYSTEM FOR DOCUMENT PREPARATION

P. Chartier and F. Blais

ERB-983

NRCC No. 25786

MAY 1986

TABLE OF CONTENTS

	Page
ABSTRACT / RÉSUMÉ	v
INTRODUCTION	1
THE HARDWARE CONFIGURATION	1
THE SOFTWARE LAYERS	2
SPECIFIC COMPONENTS	3
Digitization	3
Font Formatting	4
Using The Features	6
CONCLUSION	8
REFERENCES	8
APPENDIX 1	9

ABSTRACT

This report describes a set of hardware and software components enabling the automatic, semi-automatic or manual generation of bit map images. These can be downloaded to an output device, and used as additional fonts, Logos, etc. It includes a simple typesetting system adapted to the fonts which are produced. The key features of the system are its ease of use and overall simplicity. The implementation is exemplified using UNIX™ as a development environment, LEX, and the C programming language. The hardware used is an image processor, a low cost CCD camera, and a microprocessor. The target output device was a QMS-800 laser printer.

The system includes software to
- digitize fonts
- format characters
- store them in a user-alterable format
- translate the above format into printer-specific commands
- select and control the printing

Although this system was not aimed at generality, interested practitioners will immediately see how to adapt this base system to their own situation and equipment configuration.

RÉSUMÉ

Le présent rapport décrit un jeu de logiciels facilitant la génération automatique, semi-automatique ou manuelle d'images en tableaux de points. Ceux-cis peuvent être envoyés à un périphérique de sortie pour être utilisés comme fontes supplémentaires, logos ou autres. Le logiciel inclue aussi un programme de mise en pages facile à utiliser. Les points forts du système sont la simplicité et la facilité d'utilisation. Le système fonctionne sous UNIX™, utilise LEX et le langage "C". Le matériel utilisé comprends un processeur d'images, une caméra CCD et un microprocesseur. Le périphérique d'impression est un QMS-800 laser printer.

Le système comprends tout le logiciel pour:
- numériser les fontes
- conformer les caractères
- les emmagasiner sur fichier
- traduire le contenu de ces fichiers pour l'imprimante
- contrôler l'impression

Même si ce système ne se veut pas parfaitement général et flexible, les usagers potentiels seront à même de constater comme il est facile de l'adapter à des besoins spécifiques.

A SEMI-AUTOMATIC SYSTEM FOR DOCUMENT PREPARATION

P. Chartier and F. Blais

INTRODUCTION

The advent of highly adaptable output media, specifically laser printers, creates a demand for a variety of fonts, patterns, and special symbols, which a user or service provider could easily create, down-load, and incorporate in its production.

We propose, here, the basis of a system that draws on standard resources to create font tables, without resorting to a special font editor, but uses a regular text editor to specify bit map data to be down-loaded as font tables for a QMS-800 laser printer or similar devices. The inconveniences associated with the rectangular aspect ratio of normal terminals are alleviated by another sub-system which enables the digitizing of the individual characters and the conversion to the file format just mentioned. This allows input of new fonts at much faster rates.

The complete system, such as that used for this demonstration, is capable of much more than described here, but is representative of a low-cost, off-the-shelf image processing system. We will describe only in broad terms the image processor, its host CPU, and the camera, since they are fairly standard in functions and capabilities.

We will describe the digitizer's operation, the various parts of the control program. We will then describe the file format, its keywords, and potential options. We will then examine the program that decodes this file and compiles it in a series of commands for the QMS-800 laser printer.

The last part of this dissertation will describe a filter that interprets a number of page formatting commands in order to change fonts, set margins, or simulate partial line feeds to print subscripts or exponents, and some more.

This system is not a commercial product, rather an experimental one, but is easily duplicated, understood, and expanded. The software is easily adapted to any system that already contains the basic peripherals described in the first part of this report. It also automates a job that remains tedious, at best, when done on even the better font editors.

THE HARDWARE CONFIGURATION

The processor used for this demonstration is a COSMOS SYSTEM's LYRA 20 microprocessor. The machine is based on a Motorola 68000 chip, in the super-micro class. The card-cage of the LYRA 20 contains a MULTIBUS backplane, which opens the door to multiple hardware additions. One of those is a MATROX frame grabber and an image display sub-system. The frame grabber is connected to a black and white CCD camera.

Many suppliers offer complete image processing and digitization systems based

on similar configurations. A user already equipped with such a system needs only software, to be able to generate fonts himself. The printer is connected to the neighbouring VAX 11/750 system. The connection between the two machines (both running under UNIX™) is made through RS-232 type links, and the printer itself looks as a mere terminal to the VAX. The VAX, in this configuration, is simply a data relay between the micro and the printer and is there only because of historical reasons. The printer could be connected to the micro itself, only efficiency would be gained, and nothing lost, as far as the system we describe is concerned. The only component that is not a standard part of a regular data processing installation is the image processing sub-system.

MATROX, a Montreal based manufacturer of electronic components, produces the VAF-512 and the RGB-GRAPH boards. Those cards are under the direct control of the main processor. No special drivers have been added to UNIX™ to interface with those, but operation must be carried out in UNIX™'s super-user mode. Those graphic boards receive their input from a SONY CCD camera of the XC-37 type. This camera is highly linear and has ample resolution for the purpose described here.

THE SOFTWARE LAYERS

The software needed to operate the camera has been constructed in successive layers. Each layer is an arbitrary logical separation defined by the functions of the modules. There are six so-called primitive layers. These control functions further and further remote from the basic hardware contingencies. They are summarised here:

Level 0	Direct hardware control
Level 1	Line and dot generation/ cursor control
Level 2	Basic geometric primitives
Level 3	Alphanumeric display
Level 4	Screen independant display with clipping
Level 5	Diagram and chart plotting utilities

These primitives form the backbone of the utilities developed for image processing, data acquisition, and display manipulation. Of those numerous functions, developed for a robotic vision project, only a minute fraction were needed. The basic operation is digitization; the rest of the process is carried out using basic graphic manipulations. The process is embodied in a program that takes care of user interaction, display of the font, and storage of data on disk. The data captured by the system are processed by the program to produce a text file that represents the bit map information as strings of blanks and ones ("1") in a text file.

The text file is then sent to a filter program that separates the bit map information from other ancillary data and translates the whole contents of the file into a format that the printer can readily use.

The format of the file makes it readily alterable using any text editor, or transferable as ASCII text. Those features are highly useful when it is necessary to add to a character, "polish" it by hand, or merge a number of fonts together.

The last phase involves using the characters in a given text. The end product

of the program is a text file representing a bit map of the pixels needed to generate the individual characters. In our demonstration, it is coded in the QUIC language, specific to the QMS-800 laser printer.

The micro is running under the UNIX™ operating system. All the software described here has been developed using either C language, or C subroutines imbedded in LEX programs. LEX itself is a program that generates user specified lexical analysers. This program provides a powerful leverage to programmers' productivity when it is necessary to define a set of key-words representing a language to be interpreted. An example of a partial LEX specification for a regular grammar is given in APPENDIX 1.

A program named **deffont** has been generated using LEX in order to do the lexical analysis. Since LEX generates a compiler for regular grammars only, the command language for the font description file had to limit itself to a simple <keyword> <parameter> form.

SPECIFIC COMPONENTS

Three programs were created specifically for this demonstration. They perform the automatic character imaging, the font description table translation, and the limited typesetting command language translation.

We will refer to them by their names, **fontecam**, **deffont**, and **accents**, respectively. Only **fontecam** has been completely written in "C". The other two are LEX specifications for a regular grammar.

Digitization

The **fontecam** program is a menu driven user interface. It is composed of seven main blocks representing the necessary steps for production of a font description table. The visible portion of the program, the menu, relies on a set of functions supplied with the UNIX™ system, **curses**. These functions manage the cursor movement and input of data on a conventional terminal.

The main components' functions are

- Specification - dimension of the characters' reference frame
  - camera gain and image thresholding
  - search area
  - margins
- Capture - freezes the image into the memory
- Search - retrieves the character's exact position and isolates it from its surroundings

- Centering - position the image manually within a reference frame

- Filtered mapping
  - scales down and transfers the original image into the desired font grid
  - filters out aliasing and reduction errors
  - plot the resulting image for evaluation
  - zooms the resulting image

- Saving - translates the internal bit-map representation of the character into a font definition table (FDT)
  - stores FDT on disk

The desired character is placed in front of the camera, digitized, and the search area is identified on the graphic output. The user has the possibility of defining fixed font format, in which case the image will be mapped into the font grid. He can also select proportional format, where the width of the font will be adjusted relative to a fixed height.

The rule governing the mapping is that the original character's window will be split into a number of rectangles equal in number to the number of squares found in the target font grid. There is, then, a correspondance established between the source character and the target character. Each square in the target font grid will correspond to a pixel on the output device. A pixel in the grid will be turned on when the surface of the correponding rectangle of the source character contains a sufficient number of pixels that are "on". This "sufficient number" is defined by the user, as well as what gray levels correspond to the "on" state.

Because of the mapping rules, which would cause a comma (,) to fill the same area as the upper-case M, the user sometimes has to adjust the positon of the character by hand. A menu command is provided for doing just that, allowing the user to interact with the image with graphic cursors. When he is satisfied with what he sees, the operator can request the saving of the character grid contents onto the disk, as a character file. The command to do so allows also an optional zooming factor to allow the user to capture a number of fonts for each digitization.

This is useful when the output device cannot perform scaling on its font tables, as is the case with the QMS-800 laser printer and many other output devices. Ultimately, the character on the screen will reach an acceptable appearance, and the user can save it. The data format is that of a conventional text file.

Font Formatting

The font definition file contains information allowing the printer to control character position within its definition window. This information is specified by associating it to various keywords. They are:

* debut Initializes the program and signals that FDT follows.

* font nnn	Declares the following as font #nnn
* hg nn	Sets the glyph height to nn pixels.
char HH	The following bit map data defines character #HH; HH is the ASCII number of the character, in HEXA-DECIMAL.
wd+ +	Sets the following characters' width to the distance between the two "+" signs ("+" included).
bm	The line represents bit map data, 1 represents "on" dot, space represents "off" dot. Trailing spaces need not be there at all, excess data are truncated in line with the preceding "wd" command.
un autre	Ends current character, starts another one.
fini	End of Font Definition Table. Any subsequent data are printed as normal text.

* These must be defined before the first "char" command.

There are many other parameters that could be set to create bit maps with more subtlety. They were preset to their default values, to simplify creation of bit map font definition file. Thus, the above commands represent the minimum level of complexity one has to surmount to create or add to a font.

The font number can be that of an existing font. In this case, characters will be added to that font, or will redefine existing characters, if the argument of the "char" command corresponds to the ASCII code of that existing character. A font definition table can be generated entirely manually, using a text editor, automatically using the camera, or altered in any manner a normal text file can be with an editor.

The following command will cause the translation of the QMS-800 laser printer supplementary characters for font 104:

deffont <f104.sup >f104.prn

The output file **f104.prn** is also a text file, and it can be altered using a standard text editor. The format and contents of the file, though, are far less becoming than that of the FDT it comes from. The contents is also directly dependant on the machine used as a printer, whereas the FDT can be translated to accomodate different printers or even terminals which have downloadable fonts.

In order to down-load the newly defined font, it is simply necessary to send it to the printer, as if it were regular text. Nothing will be printed, though. The following will do just that on UNITY™:

font f104

All you need, then, is to use the fonts in your printouts. This is best done with the **accents** program beautifier and decorator.

Using the Features

The last phase is built as a filter that recognises a number of constructs from its input text and translates them into printer-specific QUIC commands. The syntax of the commands is reminiscent of that of other text formatters, such as **nroff** and **troff** or VMS-based **DSR**. As much as possible we attempted to keep that syntax independant of that of the above mentioned formatters, but the outcome leaves a bit to be desired.

There are conflicts; **accents** is thus best used alone, for short texts or texts that do not require the page formatting power of a standard formatter. The conflicts with **DSR** are very easy to resolve, though, and one can use **accents** as a post-processor on files formatted with it. On UNITY™, files generated under VMS have been processed without any problem by **accents**. It must be pointed out, though, that this report has been processed in its entirety with **accents** and no other preprocessor has been used.

The main purpose of the program was to process French text, translating accented letters, represented as combinations of characters, into special codes to cause the printer to use the accented character from its supplementary font. As an example, the quote (') represents the acute accent in the following:

```
e'   e%'  e\b'
```

here, the \b represents a backspace, and any of these will print as:

é

in this system, the following are defined:

é è ê ë à â î ï ô ü û ù

é è ê ë à â î ï ô ü û ù

é è ê ë à â î ï ô ü û ù

é è ê ë à â î ï ô ü û ù

é è ê ë à â î ï ô ü û ù

é è ê ë à â î ï ô ü û ù

There are other fonts, like gothic (Old English), that have been digitized using the system. They are **oe1000.prn** and **oe1001.prn** .

oe1000.prn is a fairly good looking
font, specially suited for mathematical
essays related to pre-Newtonian theories.

OE1001 PRN IS BEST SUITED FOR NEAR SIGHTED SCHOLARS

The program will also recognise commands to perform partial or multiple line feeds, align top and bottom of page, margins, line and character spacing, and also change fonts on the fly.

This *is an example* of **what** can be done.

T_{h_s} also is an example.

To be able to change fonts, one must first declare it:

\font 4 114

This declares built-in font #104 as number 4. It also makes that font the default for the following text, until changed with a '%' flag or another \font declaration. The '%' flag can be used within a line, not \font.

To use it, one uses %4 in front of the text that is to be printed with that font. To terminate, %0 represents whatever font was used before the last font change.

To perform partial line feeds, the (^) and (v) are to be used. The default distance has been set to 0.050 inches. To change that distance, the command:

\saut n.nn

will cause the (^) and (v) to move by n.nn inches up or down .

\saut

without arguments, will return to the default distance of 0.050 ins. The commands beginning with a \ must all start on the first column, and be alone on their line along with their parameters.

The command (=++) will set the printer to two lines per inch, one line per inch for each '+' sign. (=++++++) gives six lines per inch, which is an acceptable

line density for most work. In both cases omission of the '+' sign reverts the printer to its default settings. For character spacing, (l) gives proportional spacing.

The major feature of this filter is that being constructed as a lexical analyser, using LEX, it is easy to modify and quick to construct. One simply has to enumerate the constructs that are to be interpreted, describing them as regular expressions, and specify what to do using C expressions. These can be placed on the same line as the regular expression, or they can be appended at the end. Appendix 1 gives the complete source for **accent.lex** which is the analysis part of a shell script, **accents** (note the plural) that also sends the result to the QMS-800 laser printer.

CONCLUSION

The various elements described here are certainly far from the performances or functionality a professional requires for commercial production. They nevertheless represent the foundation of a powerful system, yet an affordable one. The system can be used and extended to recognise and translate special character sequences and translate them into one special character. Foreign languages like Greek or Arabic have special glyphs for some terminal letters; these can be identified as such by simple constructs and replaced properly. Finally, the construction and maintenance of filters made using a tool like LEX, is extremely fast and straightforward.

Even if it is not a complete system, by far, it is powerful enough to serve the needs of a small production enterprise, and flexible enough to be custom tailored and completed.

REFERENCES

Lesk,M.E. and Schmidt, E. *Lex - A Lexical Analyser Generator,*
Bell Laboratories, Murray Hill New Jersey 07974

Quality Micro Systems *QUIC - Programming Language Version 3,*
P.N. 1720460.0100, Mobile, Alabama Al 36689

APPENDIX 1

```
w                       [wW]
x                       [xX]
y                       [yY]
z                       [zZ]
number                  [0-9]
integer         number+
real            number+?\.?number*
hexdigit                [0-9A-Fa-f]
nl                      [\n\r]
integ_args      ([ \t\+\-]+integer)*[ \t]*nl
real_args       ([ \t\+\-]+real)*[ \t]*nl
%START ON_JUS ON_A OFF_A CH_TAB
%k 4000
%e 4000
%p 4000
%%
<OFF_A>.                ECHO;
<OFF_A>[\n\r]           ECHO;
<ON_A>\\                raw_character(yytext[1]);
<ON_A>"\^"              raw_character("^");
<ON_A>"^"                       raw_character("^");raw_character("^");
<ON_A>\\[\n\r]          ;

^\\accent[ \t]*[\r\n]   accn();
                                                unput(yytext[yyleng-1]);
<ON_A>^\\no[ \t]*accent[ \t]*[\r\n]     noaccn();
                                                        BEGIN OFF_A;
                                                unput(yytext[yyleng-1]);

<ON_A>e%\'e\'   accen_letter(AIGU);     another_char();
<ON_A>e%\`e\'   accen_letter(GRAVE);another_char();
<ON_A>e%\"e\"   accen_letter(TREMA);another_char();
<ON_A>e%\^e\^   accen_letter(CIRCONFLEXE);another_char();
```

Sample LEX specification for part of the text formatter